

UNCLASSIFIED

DRES



DEFENCE RESEARCH ESTABLISHMENT SUFFIELD

CR 98-25

UNCLASSIFIED

Modelling of In-Soil Explosions and Interactions with Structures

BY:

Combustion Dynamics Ltd.

**SUBMITTED TO:
SCIENTIFIC AUTHORITY**

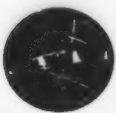
**Denis Bergeron
Defence Research Establishment Suffield
Box 4000, Medicine Hat, Alberta, T1A 8K6**

April 1998

**Controlled by Source
(Contains Company "Sensitive" Information)**

WARNING

"The use of this information is permitted subject to recognition of proprietary and patent rights."



CRAD



National Defence

Défense nationale

UNCLASSIFIED

Canada

102-02548/



Combustion Dynamics Ltd.



Modelling of In-Soil Explosions and Interactions with Structures

**Report - Requirements Document - Design Documents
for Amendment of SSC Contract
W7702-5-R548**

**Combustion Dynamics Ltd.
Suite 203, 132 4th Ave. S.E.
Medicine Hat, Alberta
Canada T1A 8B5**

April 20, 1998

Combustion Dynamics Ltd.

Introduction

This report contains three main items:

- 1. The requirements document of the IFSAS-DYNA link for fluid-structure interaction (FSI) coupling.**
- 2. The design documents of the IFSAS-DYNA link. This basically explains the philosophy used to construct the link, and identifies the tasks necessary for implementation.**
- 3. The report outlining the testing performed to validate the performance of the link.**

**Requirements Document for Coupling
IFSAS and LSDYNA-3D for MRL and
DRES**

by

**Rick Link
Combustion Dynamics
Revision 1
04/03/97**

TABLE OF CONTENTS

1.0 Introduction	1
2.0 Basic Requirements	2
2.1 Data Transfer	2
2.2 Speed	3
2.3 AGS	4
2.4 Geometry Model Translator	4
2.5 Driver Program	4
3.0 Tentative Task Listing	7

1.0 Introduction

This document outlines the necessary requirements to couple the current CFD code with the explicit finite element code LSDYNA-3D to obtain an effective fluid structure interaction solution technique. These requirements exist as a result of proposals to MRL and DRES, which identified the solution of the following problem categories:

- 1) Vulnerability of structures comprised of thin structural members (plates and shells) to a variety of explosive loadings.
- 2) Wave transmission through thick solid structural objects due to explosive loading.

With the above problems in mind, it was proposed that the following LSDYNA-3D elements would be coupled with the current CFD code:

- 1) 3-D shell elements
- 2) 3-D solid elements

These elements would be coupled in both the FCT and Godunov AGS CFD codes. The coupling would be implemented on the DEC-Alpha and SGI parallel platforms, and on the HP non-parallel platform.

Before outlining the project requirements, the assumptions made to simplify the project scope are as follows:

- 1) The CFD and structural models are generated in separate GUI's.
- 2) Material rupture i.e. formation of new fracture surfaces is not included in the CFD models.
- 3) Access to the DYNA source code is possible.
- 4) Only 4 node shell and 8 noded brick elements are considered in the FSI models, although other elements (such as beams or stiffeners) may be connected to these main elements. They are, however, ignored in FSI calculations.
- 5) The basic formulation adopted for the FSI coupling is the coupled Lagrangian-Eulerian formulation which was used to couple IFSAS with VAST shell elements.

2.0 Basic Requirements

Figure 2.1 shows the basic flow of the analysis procedure. Initially, the solid geometric model is generated by LSDYNA. This model is then translated to input for the CFD module, where the fluid geometry is then created. The FSI calculation is then run, and output is generated for both CFD and LSDYNA modules.

Based on the flowchart in Figure 2.1, the general requirements for this project are as follows:

- 1) Implementation of an efficient data transfer mechanism between the CFD and LSDYNA codes - problem calculation segment. The data transfer must work on both parallel and non-parallel machines.
- 2) Increase in calculation speed in the CFD module - problem calculation segment. This was identified as a major problem from calculations performed on large finite element meshes.
- 3) Implementation of slanted surface algorithms in the AGS code for general finite element meshes - calculation segment. Although completed in the FCT code, this is not done for the Godunov AGS code. This is necessary in order to obtain increased resolution of the blast front, thereby increasing peak pressure and impulse accuracy on the structure.
- 4) Implementation of geometry model creator in the CFD module - between solid geometry and fluid geometry creation. Solid models must be translated to input for the CFD module.
- 5) Implementation of a driver program to control information flow - calculation segment. It is desired to maintain IFSAS and LSDYNA as two separate codes. This will require a driver program to control the flow of information, as done for IFSAS and VAST.

2.1 Data Transfer

Figure 2.2 shows the basic flow of the Lagrangian-Eulerian FSI formulation. The fluids problem is solved first, providing pressures on the structural surface. These pressures are used by the structural solver to compute velocities and displacements, which are used as boundary conditions for the fluids solver.

There are two possible methods of connection; through disk files, and through RAM. Before deciding on one or the other, the pros and cons of each should be evaluated.

Disk Transfer - this is the method currently in use. This is performed through the use of two files, the Neutral Pressure Loading File (NPLF), and the Neutral Displacement and Velocity File (NDVF), the meanings of which are obvious. The advantages of disk transfer are as follows

- 1) Maintenance of separate codes - each code is restarted as required. This minimizes the changes that are necessary to each code, since each is a "black box".
- 2) RAM and processor use minimization - since only one process is running at a time, both processes do not need to be active at the same time.

The main disadvantage of disk transfer is the overhead required for restarting each calculation.

Each process must read in a restart file. There was some suggestion in the proposal that this was the cause for slow execution times in some runs with VAST. However, this requires verification; the slow times are probably more due with the way that the structural finite elements are stored in the CFD database (discussed later).

RAM Transfer - this involves using shared memory to transfer information between the two codes. This is allegedly performed using MPI (don't know the abbreviation, you guys have to elaborate). The main advantage to using this method is that each process stays in memory, and therefore restart information does not have to be read. Therefore, the code becomes faster due to the elimination of restart overhead. The main disadvantages are as follows:

- 1) Increased RAM requirements - each process commands its share of RAM, plus there is RAM associated with data transfer. This would double the amount of RAM required in the worst case.
- 2) New implementation - this has not been done before, and there is some doubt whether this can be done at all for different programming languages (FORTRAN and C). The last time we tried something like this, it was not pretty.
- 3) Memory blocking - some finite element code use out of core storage for large problems. If LSDYNA uses this (and I bet they do), the job of transferring data becomes significantly harder to accomplish.

2.2 Speed

As mentioned previously, slow execution times obtained for large 3-D calculations with VAST warrants an examination of the speed problem of the existing code. Possible reasons include the following:

- 1) Database management - currently, I am under the impression that large finite element surfaces are stored as separate elements. Each element is stored as a separate database record. Previous experience in saving 2-D solid elements in the database (2-D IFSAS is done nearly identically to 3-D IFSAS-VAST) indicates that large numbers of elements requires substantially more time than small numbers. However, large CFD grid sizes seem to save very quickly. This would indicate to me that we need to store these elements in a more efficient manner. A possible solution would be to store all node information in a separate record, and all element information in a separate record. The finite element mesh can then be generated when this information is read. This may save a substantial portion of time.
- 2) Surface detection - I am unaware of how the surface detection algorithms work. There may be a way to optimize the speed of these algorithms if we know how they work. A clear document needs to be written to explain the surface detection algorithms for NGF shapes. This is then reviewed by technical staff, which recommend possible changes to optimize speed.
- 3) Restart facilities - some overhead is incurred by reading in restart files. However, to blame this for slow execution times without testing it can be misleading. Large problems should be run and restarted to determine the effect of this overhead on overall execution times.

2.3 AGS

The Adaptive Gridding Scheme (AGS) currently in Version 1.6 does not support the input of solid geometry files. The capability to support 3-D solid geometry files for large displacements (> 1 CFD cell) must be implemented.

2.4 Geometry Model Translator

Currently, Neutral Geometry File (NGF) format is translated to input for the CFD module. The creation of the NGF is done using a pre-processing feature in VAST Graphics (VG), developed by Martec. A similar geometry translator must be written for LSDYNA, and can be written similar to the Martec one in IGES format.

It is noted here that 3-D solid elements are treated no differently than shell elements. The CFD code only requires exterior surfaces - surfaces where pressure is specified. This means that LSDYNA 3-D pressure surfaces are used as "shells" to define the structure. However, "thin" solid objects (thickness < 1 CFD cell) will have to be accommodated, similar to the way 2-D thin solids are treated.

2.5 Driver Program

This program controls the flow of information between the two codes. The driver code must do the following tasks:

- 1) Read input data including
 - a) Time information - analysis start and end times
 - b) File information for LSDYNA - all input files required
 - c) Project and run number for CFD code
 - d) Any other files if NPLF or NDVF files are used
- 2) Handle communication between the two codes - if done by shared memory, MPI would be used here. If done by files, the NPLF and NDVF files would be used.
- 3) Handle time stepping - this would be automatic. The critical time step would be obtained for both the CFD and structural calculations. The largest of the two would serve as a loop for the smaller one.

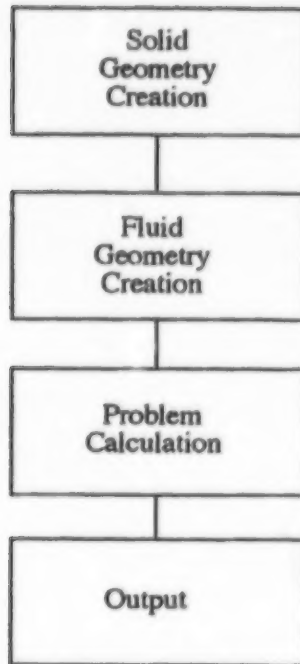


Figure 2.1 Analysis flowchart

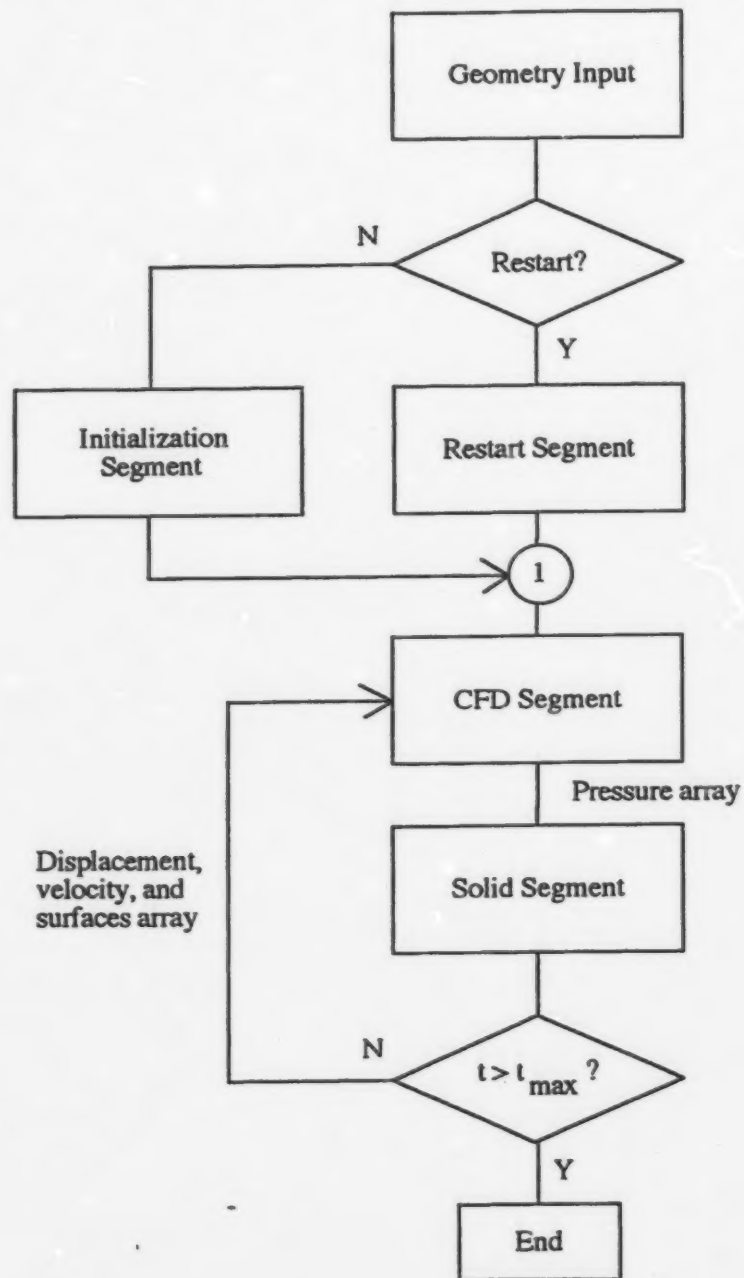


Figure 2.2 Calculation segment flowchart

3.0 Tentative Task Listing

Figure 3.1 is a tentative flowchart for the project. It is divided into two phases: the 3-D shell phase (before May 31), and the 3-D solid phase (after May 31). Task breakdowns are as follows:

Shell Phase - Tasks 1-7

Task 1 - Requirements Document - RL, PT

- a) Determine requirements for the project.
- b) Determine task breakdown in order to clearly define the work.

Task 2 - Design Document - RL, DW, PT, JP, LC, DT, FZ

- a) From requirements document and input from involved people, write design document which covers all tasks of project. This work may be split among all people.

Task 3 - Speed/Data Transfer - LC, GK, JP

- a) Find out why CFD code (version 1.5) is so slow in initializing large finite element problems. Use VAST models to test.
- b) Correct problems associated with speed slowdown, and test using VAST models.
- c) Move corrected code to version 1.6, and retest.
- d) If common shared memory is to be used, implement memory scheme using MPI on DRES DEC Alpha for non-adaptive hydrocode. Real VAST or fake VAST will be used for the testing.

Task 4 - AGS Implementation - DT, FZ, LC

- a) Implement NGF import and calculation into AGS code and test. VAST models will be used for testing.
- b) Step b) of Task 3.
- c) Step d) of Task 3.

Task 5 - DYNA Type Element - RL

- a) Obtain information regarding LSDYNA input and output data.
- b) Construct a "fake" DYNA which is as similar as possible to the real LSDYNA. This is to simulate LSDYNA without having it resident here. Options include implementing a real shell element, so that we can use it in our code.

Task 6 - Driver Program - LC

- a) Construct a script program that runs the integrated FSI analysis, both on parallel and non-parallel machines.

Task 7 - Proof of Concept Calculation - RL, DW

- a) Run a calculation using the DYNA type element developed in Task 5. This could vary from a simulated deflection pattern on a plate structure to a real experiment if a real shell element is implemented.

3-D Solid Phase - Tasks 8-12

Task 8 - LSDYNA Geometry Translator - LC

- a) Obtain information regarding LSDYNA geometry description.
- b) Develop and implement program which takes LSDYNA geometry and translates it into IGES format, suitable for import into the CFD module. This includes data for 3-D shells, and 3-D pressure surfaces.

Task 9 - 3-D Solid Element - DT, JP

- a) Implement cell filling algorithm
- b) Implement thin element algorithms for 3-D solid elements.

Task 10 - DYNA Type 3-D Solid - RL

- a) Implement DYNA type 3-D solid element to the "fake" LSDYNA. This could involve putting in a real DYNA-3D solid element.

Task 11 - Testing - RL, DW

- a) Run a number of tests on 3-D shells and solids and debug.

Task 12 - LSDYNA Implementation - RL, LC

- a) Implement and test software at Livermore Software.

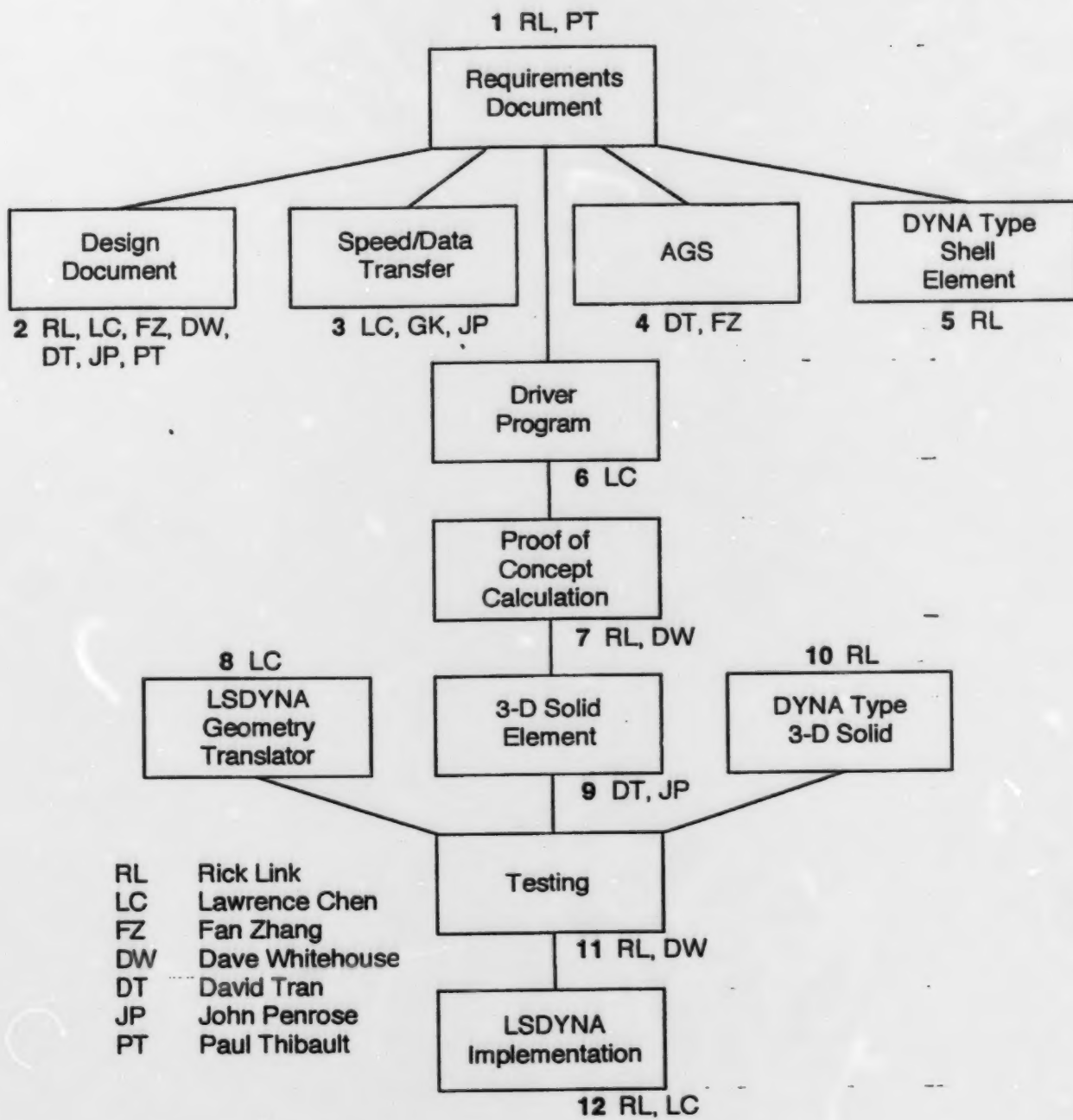


Figure 3.1 Project task flowchart

**Design Document for Coupling IFSAS
and LSDYNA-3D for MRL and DRES**

by

**Rick Link
Combustion Dynamics
Revision 2
05/08/97**

TABLE OF CONTENTS

1.0 Introduction	1
2.0 Phase 1 - Implementation of the 3-D Shell Element	4
2.1 Task 3 - Speed/Data Transfer	4
2.2 Task 4 - Implementation of AGS	5
2.3 Task 5 - DYNA Type Shell Element	6
2.4 Task 6 - Driver Program	6
3.0 Phase 2 - Implementation of the 3-D Solid Element	13
3.1 Task 8 - LSDYNA Geometry Translator	13
3.2 Task 9 - 3-D Solid Element Capabilities	14
3.3 Task 10 - 3-D Solid Element Implementation	14
4.0 References	16

1.0 Introduction

This document is meant to serve as a design guide for coupling the CFD code IFSAS with the explicit finite element code LSDYNA-3D to obtain an effective fluid-structure interaction (FSI) solution technique. The design is based upon the requirements outlined in the requirements document for this project.

Figure 1.1 shows the general philosophy behind coupling the two codes. The calculation starts on the CFD side, where pressures are calculated. The pressures are then read into the solid FEM code, where velocities and displacements are computed. These are then sent to the CFD code, and the time step is incremented. Control between the two codes is performed using a separate driver program. Advantages of using this coupling scheme include:

- 1) Minimal changes in CFD and solid codes - these codes can act as stand-alone. No implicit coupling is necessary.
- 2) No restart overhead - both codes can run at the same time, thus eliminating time overhead for restarting the codes.

Before proceeding into the coupling design, it is useful to list the general requirements used to define the project scope. These include the following:

- 1) 8-noded 3-D solid and 4-noded 3-D shells are supported.
- 2) Coupling is implemented in the Godunov AGS solver, and is implemented on the DEC-Alpha, SGI, and HP platforms.
- 3) Rezoning (remeshing) is supported.

General assumptions used here include:

- 1) The basic formulation used here is the coupled Euler-Lagrange formulation, described in the requirements document.
- 2) Material rupture is not considered here.

As outlined in the requirements document, development is divided into two main segments: implementation of 3-D shell elements and 3-D solid elements respectively. Figure 1.2 shows the proposed line of development. The 3-D shell will be implemented first using existing VAST coupling and DYNA type elements for testing. A proof of concept calculation will then be performed. Next, implementation of the 3-D solid using DYNA type elements will commence. As well, development of an LSDYNA geometry translator occurs in this phase. Finally, testing of the entire system takes place, and implementation of the code occurs at Livermore Software Technology Corporation.

This document is divided into the following sections. Section 2 describes the design features contained in implementing the 3-D shell element. Section 3 describes design features for development of the 3-D solid element.

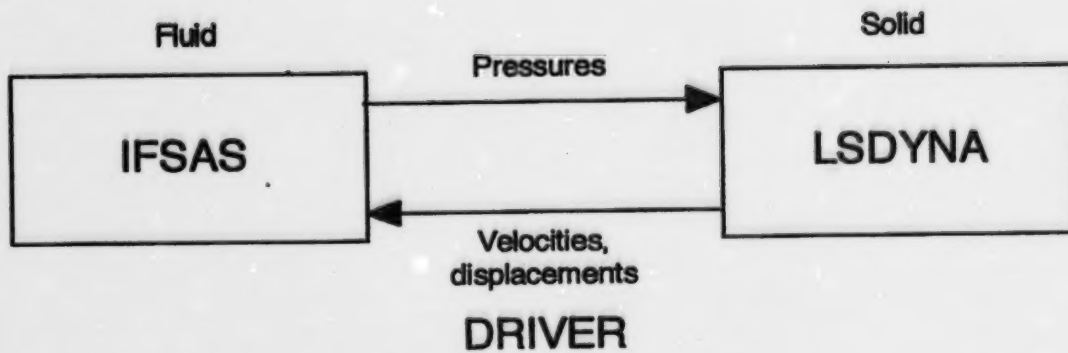


Figure 1.1 General design philosophy

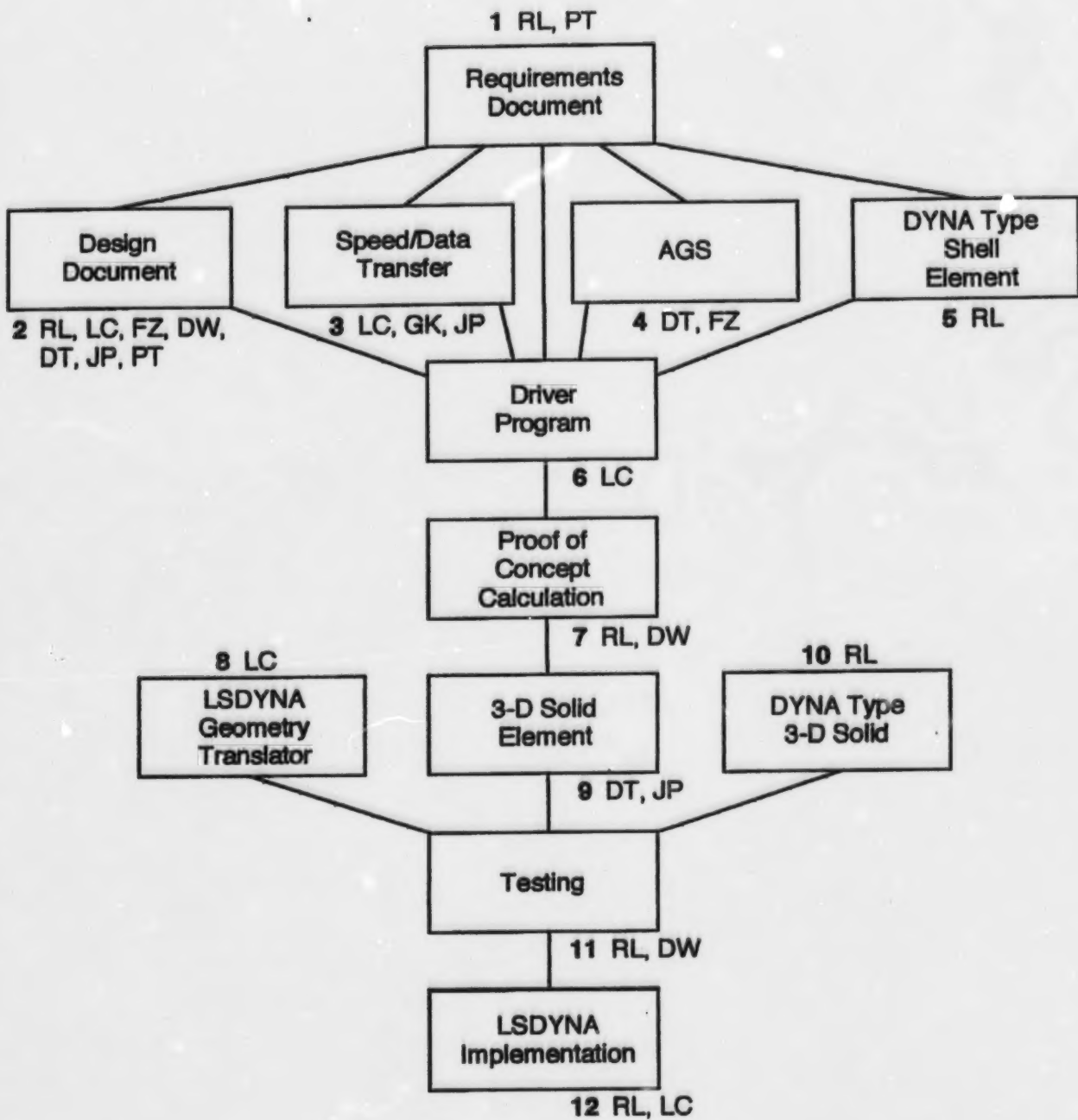


Figure 1.2 Project task flowchart

2.0 Phase 1 - Implementation of the 3-D Shell Element

The development tasks required to complete this phase are as follows:

Task 3 - Speed/Data Transfer

Task 4 - AGS Implementation

Task 5 - DYNA Type Shell Element Implementation

Task 6 - Driver Program

2.1 Task 3 - Speed/Data Transfer

This task involves the following:

- 1) Speeding up the IFSAS code for general plated shapes
- 2) Transferring data between IFSAS and LSDYNA

Previous coupling studies using VAST and IFSAS revealed that calculation times for large shell element problems is relatively slow. It seemed that the calculations ran quickly, but the time spent in initialization and data transfer was slow. Two problems were quickly found:

- a) Surface detection algorithms - these detect where the finite elements are with respect to the CFD grid. Modification of these routines resulting in a substantial speed increase.
- b) Restart algorithm - for some reason, restarts took more than twice as long as an initial time step. This was found to be the result of detecting the shape twice on the CFD grid. This was modified.

Once these changes were made, the IFSAS code was run through a routine which measured the relative times spent in each C function. The test problem was a structure composed of over 2200 triangular shell elements. The results indicated that the code used to detect pressures on the elements was taking most of the time. The reason for this was that the algorithm to detect pressures on the surface was very inefficient (an N factorial search at worst case, where N is the number of finite elements). Plans are made to modify this algorithm to an N search, which should speed up the code performance substantially. Testing will be performed using the existing VAST-IFSAS interface.

Figure 2.1 shows the basic computational setup for the FSI coupling. Since both codes are explicit, running on multiple processors is easily attained using message passing interface (MPI) language. This language is portable, and allows significant increases in speed to be attained. It is proposed here to parallelize the codes separately, since the codes are distinct entities, and it is desired to maintain them that way.

It is uncertain at this time as to how the data transfer between IFSAS and LSDYNA will occur. Since the codes are not written in the same language (IFSAS in C, LSDYNA in FORTRAN77), MPI cannot be used to perform this task. However, the following schemes are being considered.

- 1) Inter-process communication (IPC) - the transfer data would be passed using direct communication between processes. This is achieved by using IPC calls to queue messages between processes.
- 2) Data sockets - a connection is made between the two processes through a shared pipe. Data is read from and written to this common data pipe, much in the same way that data files are currently used, only this process is more direct.

Data would be transferred from C to FORTRAN using a common data format, such as XDR. Data forms will be similar to the following:

CFD to DYNA:

Pressure surface number, pressure

DYNA to CFD:

Node number, position, velocity

Pressure surface number, node 1, node 2, node 3, node 4.

Note that pressures surfaces are used instead of finite elements. This is because 3-D solid shapes contain redundant data, as described in Section 3.1.

2.2 Task 4 - Implementation of AGS

The implementation of an adaptive gridding system (AGS) is important in order to define large pressure gradients with as few CFD cells as possible. Addition of this capability requires that the existing shape detection routines must be modified. Presently, the shape detection in the CFD code works as follows:

- 1) Determine all intersecting points along each gridline - this is shown schematically in Figure 2.1. The intersecting points are used to divide the gridlines into a sequence of 1-D boundary value problems. An intersecting point is defined as a point which intersects the gridline and the shape line.
- 2) Compute boundary normals and fill computational cells
- 3) Determine the volume of cut cells - this is shown in Figure 2.3. Adjacent cell volumes are added together to form a composite cell volume.

This detection procedure must be modified because flow holes can exist using AGS. Figure 2.4 shows this for a simple case. There are no intersecting points along the line shown by the flow hole. This would result in flow being allowed through the cells. The following procedure is recommended:

- 1) Looping through the finite elements, determine which root cells (initial cells without AGS) contain each element.

- 2) Perform a search through the AGS binary tree to determine which AGS cells contain each element.
- 3) If a cell contains an element, a boundary condition is set there irregardless of whether there is an intersecting point or not.

This procedure reduces the amount of time spent in the shape detection routine, and plugs all the holes in the finite element structure.

2.3 Task 5 - DYNA Type Shell Element

The main purpose in implementing a shell element is to simulate an LSDYNA shell element as closely as possible. Keeping this in mind, one of two options will be pursued, depending on available time:

- 1) Implementation of an actual shell element - the robust shell element proposed by Bathe and Dvorkin (1986) would be implemented in a DYNA-like interface. This element would be fully integrated (no zero energy deformation modes), and would hence be much slower than the DYNA shells. In this case, it would be possible to compare proof of concept computational results with actual test results.
- 2) Implementation of a prescribed motion shell element - the geometry of the shell would be prescribed manually, and would simulated some idealized motion e.g. rigid body, sinusoidal, etc. The proof of concept calculation results would be compared with idealized fluid pressures.

2.4 Task 6 - Driver Program

The driver program performs the following tasks:

- 1) Controls the flow of data between programs.
- 2) Controls the time step size.
- 3) Acts as a watchdog so that processes initialize and exit gracefully.

Previous driver programs have been written in UNIX script language. However, it is necessary to write this program in either C or FORTRAN in order to facilitate data transfer between programs.

Before discussing the structure of the driver program, it is instructive to examine how overall FSI model generation proceeds. Figure 2.5 shows the flow of model generation. The structural model is generated first, either by manual input or by using a pre-processor such as INGRID. This generates a structural input file. The solid model is then read by a translator, which imports and translates the structural input file into a form readable by the CFD GUI. The fluids model is then constructed, and the analysis can start.

The data flow control is discussed in Section 2.1. Figures 2.6 and 2.7 show scenarios for automatic and manually controlled time stepping, respectively. The time stepping can start either from an initial time of zero, or from a non-zero restart time. The mesh can change between

restarts.

In using the automatic time stepping, it is assumed that the CFD time step for stability is larger than that generated by DYNA. This would be consistent with element sizes in the order of the CFD grid size, since sound speeds in solids greatly exceed those in fluids. LSDYNA is looped over the CFD time step. Results time dumps are performed separately in each program, and restart files are written at the end of each CFD time step. The advantage of automatic time stepping is that the user requires little knowledge of the material time scales.

Manual time stepping requires that the user input a time increment over which each program will run. Results time dumps are also performed separately in each program, and restart files are written at the end of each time increment. The advantage of manual time stepping is that the user can control the amount of restart data generated. It is possible to loop over several CFD and solid time steps.

It is recommended that both schemes be adopted in the driver program, since implementation of both are relatively simple.

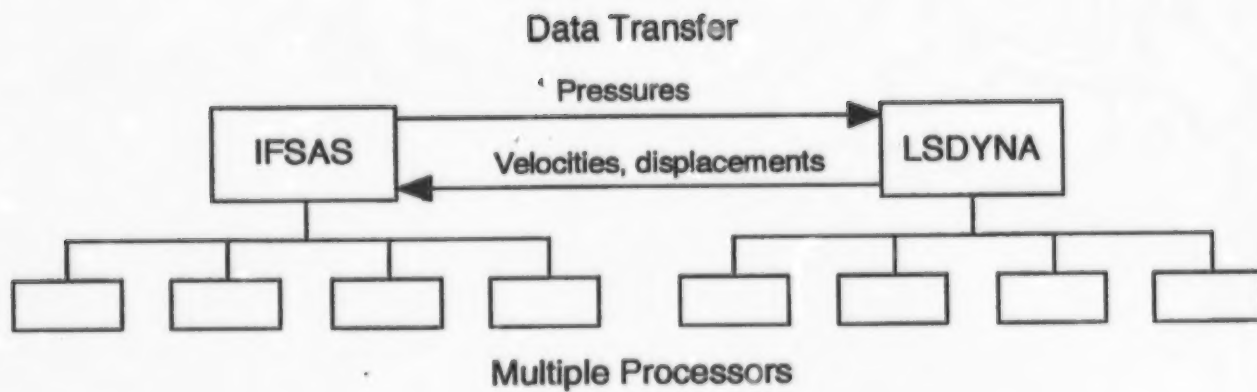


Figure 2.1 Computational setup

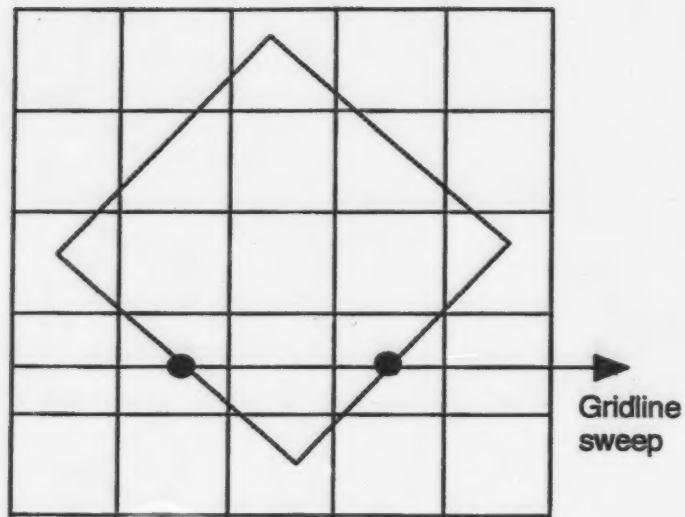


Figure 2.2 Determination of intersecting points

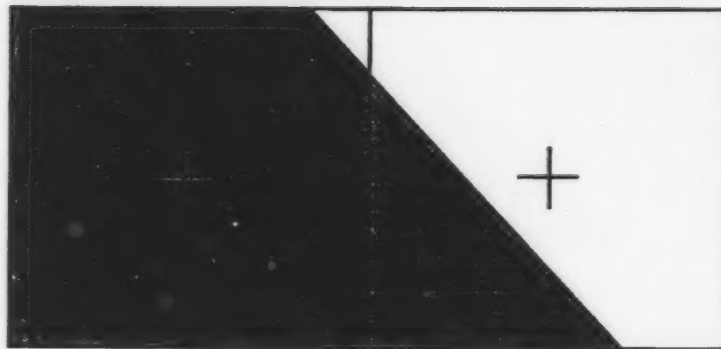


Figure 2.3 Cut cell volume

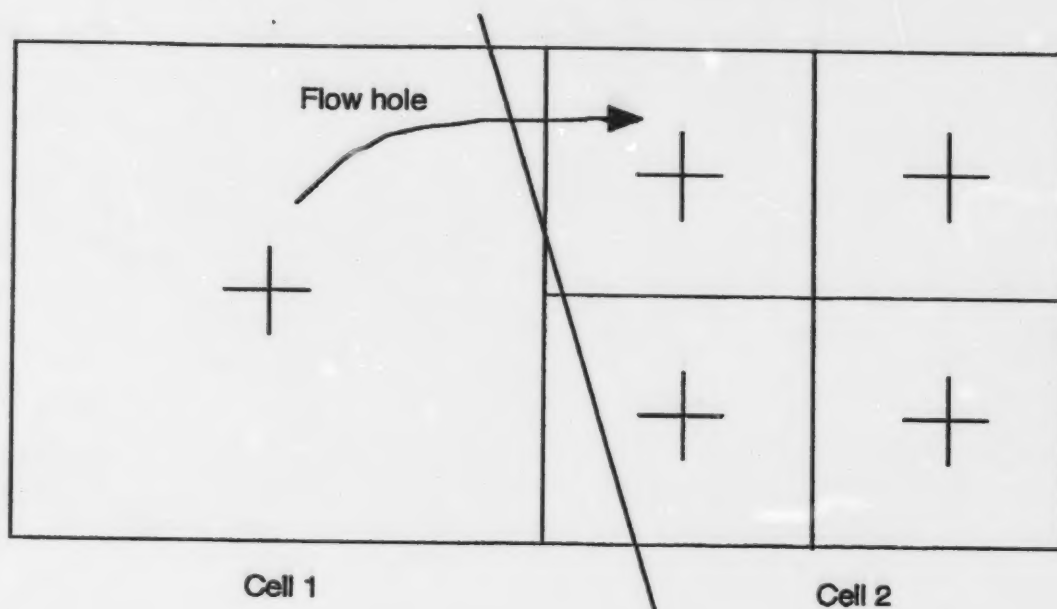


Figure 2.4 Flow hole existence

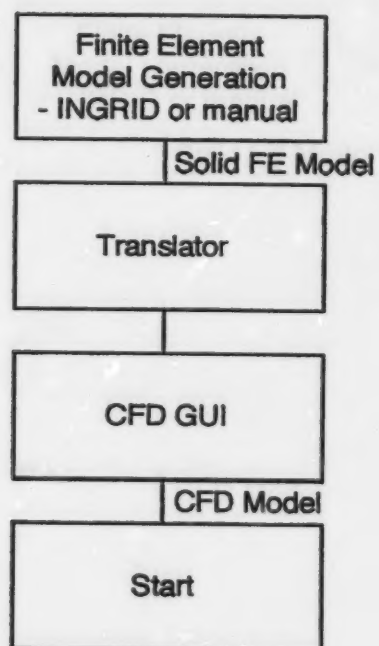


Figure 2.5 Model generation

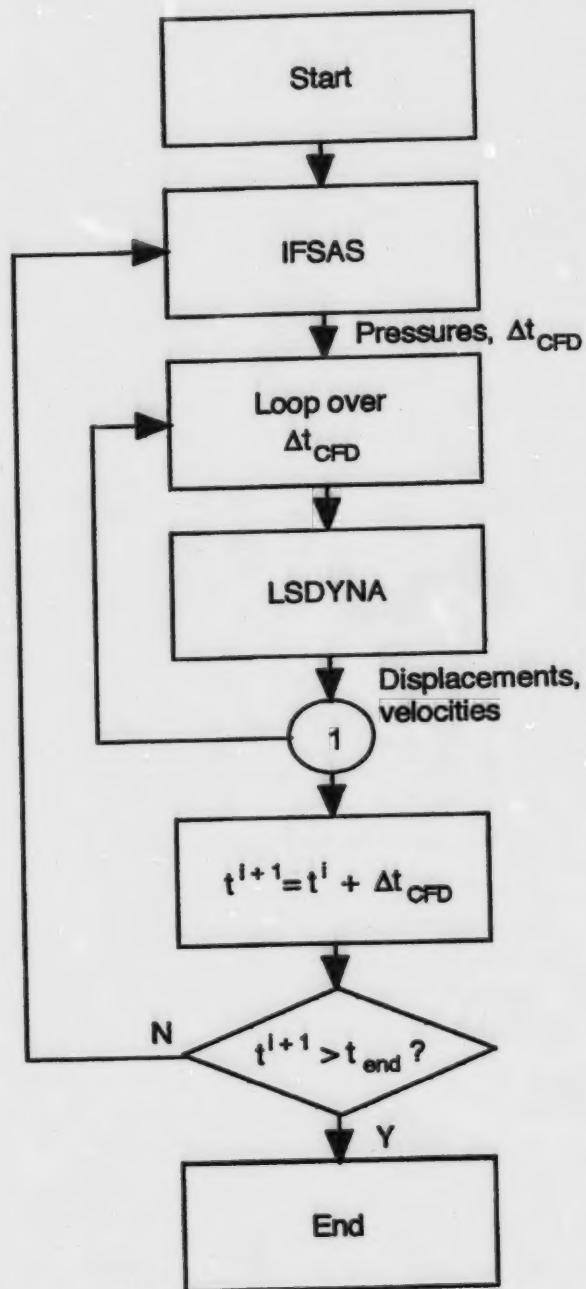


Figure 2.6 Automatic time stepping

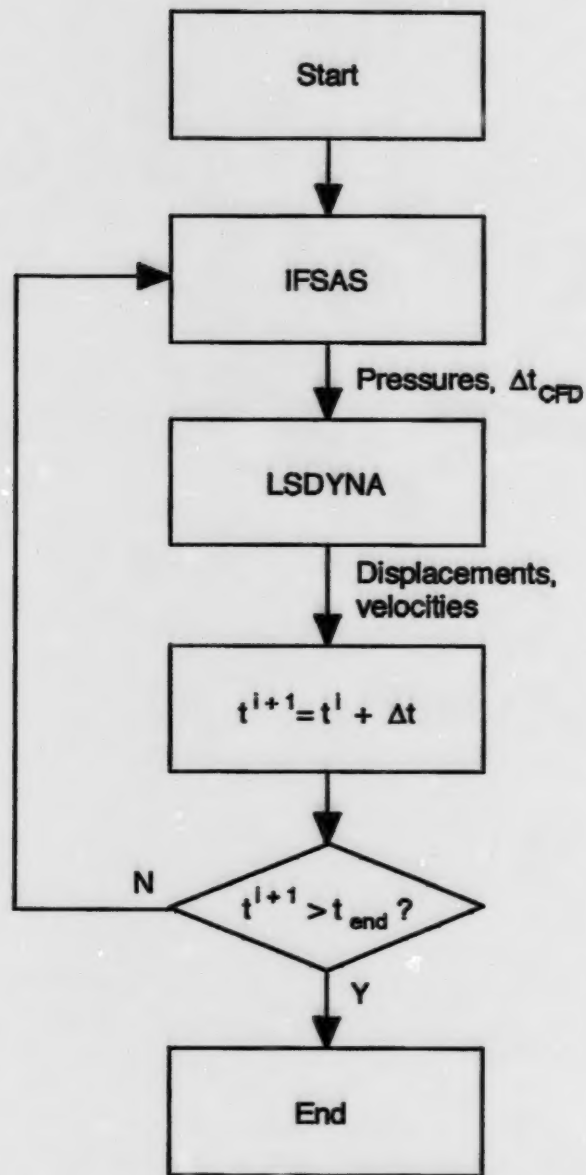


Figure 2.7 Manually controlled time stepping

3.0 Phase 2 - Implementation of the 3-D Solid Element

The development tasks required to complete this phase are as follows:

Task 8 - LSDYNA Geometry Translator

Task 9 - 3-D Solid Element Capabilities in IFSAS

Task 10 - 3-D Solid Element Implementation

3.1 Task 8 - LSDYNA Geometry Translator

As mentioned previously, the structural model must be translated into the CFD GUI in order to form the fluid model. The structural model is in the form of an input file generated by a pre-processor, or manually.

Presently, LSDYNA supports two different types of input format: standard, and keyword. The standard form supports all types of DYNA features, whereas the keyword format supports a limited amount of features. Both formats will be supported. This is because the standard format must be supported (pre-processors probably use this option), and the keyword format is easy to support (just search for the required keywords).

In creating geometry for IFSAS, the structure is treated as a number of pressure surfaces. These surfaces must be specified in the LSDYNA model. This methodology has the advantage that redundant 3-D solid data inside the solid does not have to be read.

The geometry translator will read in the following data:

- 1) Nodal coordinates
- 2) Pressure surface data
 - nodal connectivity
 - load function number

The load function number is read in so that 3-D solids and 3-D shells can be distinguished. If a load function number is not used, it is impossible to tell the difference between the two. This has significant consequences on the shape detection used in IFSAS. A 3-D shell will be identified with a load function number of 1, and a 3-D solid will assume a load function value of 2.

The current method of storing element data for the VAST finite element models has been checked for speed, and it seems adequate for large finite element models.

Previously, IGES file format has been used to generate solid models in IFSAS. It is recommended that a new file format be used that is simpler to interpret.

3.2 Task 9 - 3-D Solid Element Capabilities

The addition of 3-D solid elements will be similar to the addition of air-backed structures previously performed in IFSAS for the VAST models. If the load function number of the pressure surface is 2, the surface belongs to a 3-D solid. In the shape detection sweep, the cells in the solid are marked as solid until another 3-D solid pressure surface is encountered. At that point, the cell is marked as fluid. This is shown schematically in Figure 3.1.

3.3 Task 10 - 3-D Solid Element Implementation

As for the shell element, there are two possibilities for implementation: an actual 3-D hexahedral element, or a prescribed motion 3-D element. The actual 3-D hexahedral element used in DYNA3D (Hallquist, 1983) would be used if the first option were adopted. The element used for testing the 3-D capability of the code will largely depend on the amount of time left for implementation.

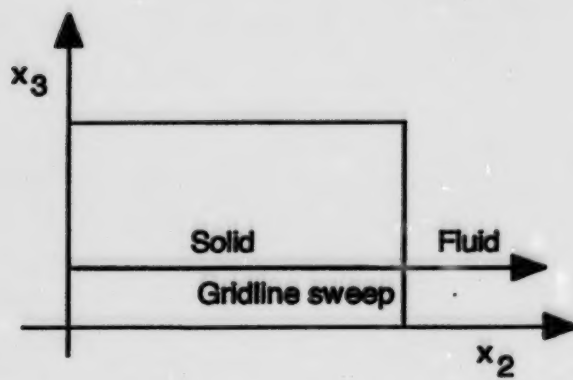
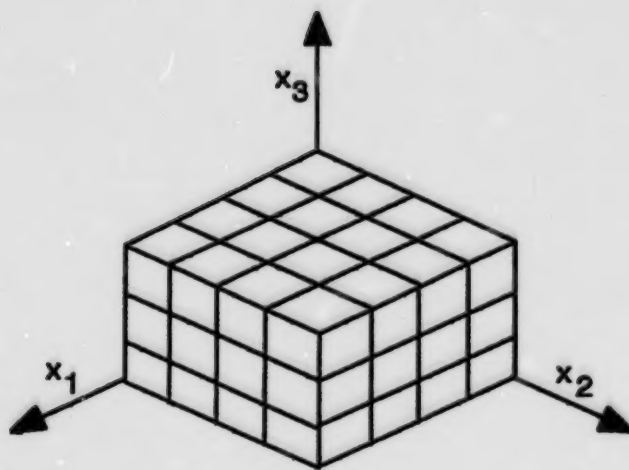


Figure 3.1 Cell marking for 3-D solid

4.0 References

- Bathe, K.J., and Dvorkin, E (1986). A Formulation of General Shell Elements - The Use of Mixed Interpolation of Tensorial Components. *International Journal for Numerical Methods in Engineering*, v. 22, pp 697-722.
- Hallquist, J.O. (1983). Theoretical Manual for DYNA3D. Lawrence Livermore National Laboratory, Report UCID 19401, 82 p.

Interim Report - IFSAS-DYNA3D Link - R. Link - 04/01/98

The IFSAS-DYNA link was tested using the following test cases:

- 1) A 2.5 m box in air subjected to 100 g TNT at the box center.
- 2) A very thin plate in air subjected to a TNT charge.

Box Test

Figure 1 shows the internal pressure seen by the box in the structural code at a time of 0.3 ms. Figure 2 shows the pressure seen at a cutting plane halfway through the box at the same time. It is seen that there is good agreement between the pressure magnitudes at the center cutting plane.

Figures 3 and 4 are pressure plots at a later time of 0.85 ms. The observed maximum pressure on the inside of the box is seen to be at the corner, which is in agreement with the CFD cutting plane results. In addition, the pressure magnitudes are similar between IFSAS and DYNA3D plots. The pressure and deformation response is seen to be symmetric, which is consistent with the explosive event scenario.

Plate Test

In order to test the large deformation capability of the shape detection in the CFD program, a very thin plate was subjected to a TNT charge. Figure 5 shows the deflection of the plate at actual scale. It is apparent that the plate is well into the large deformation range.

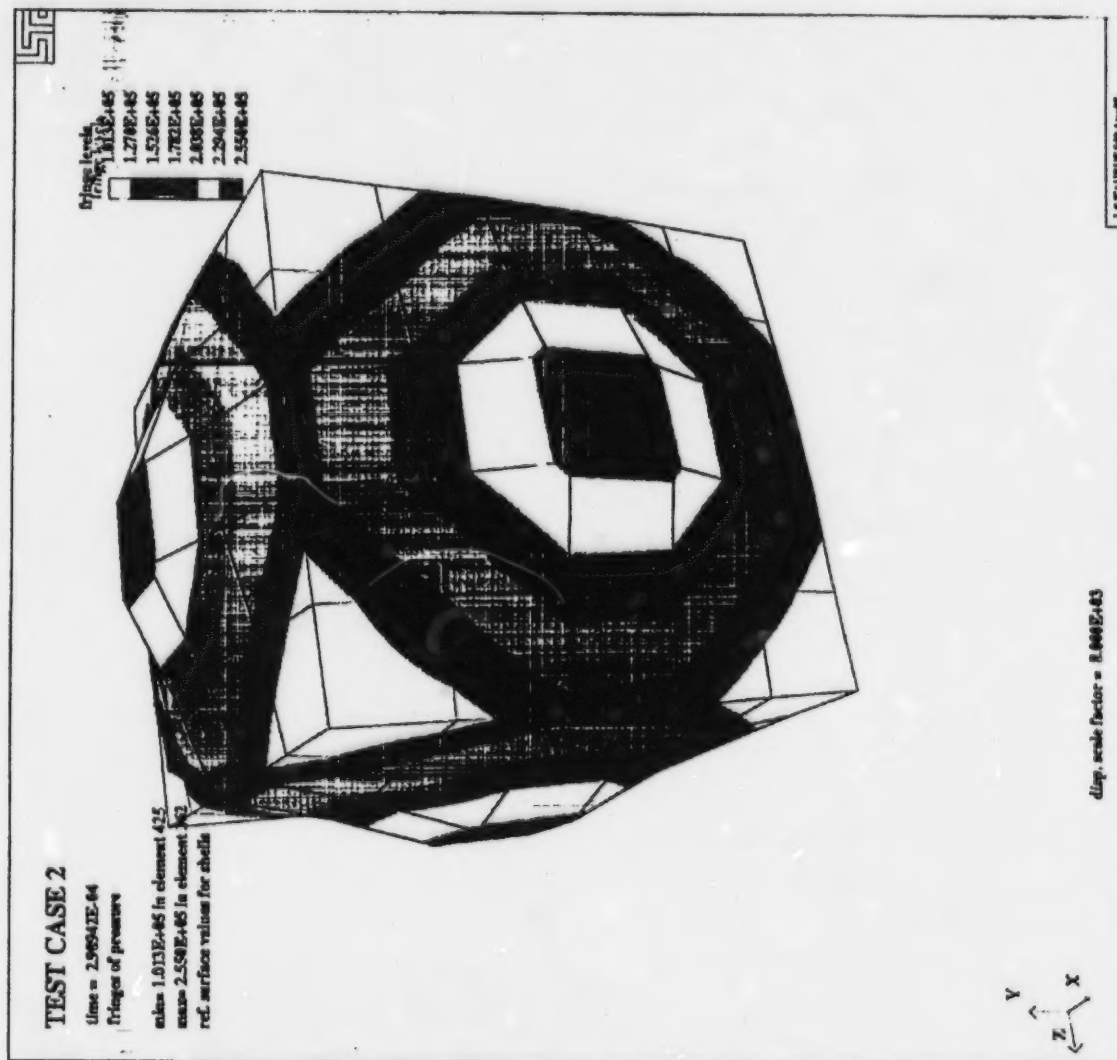
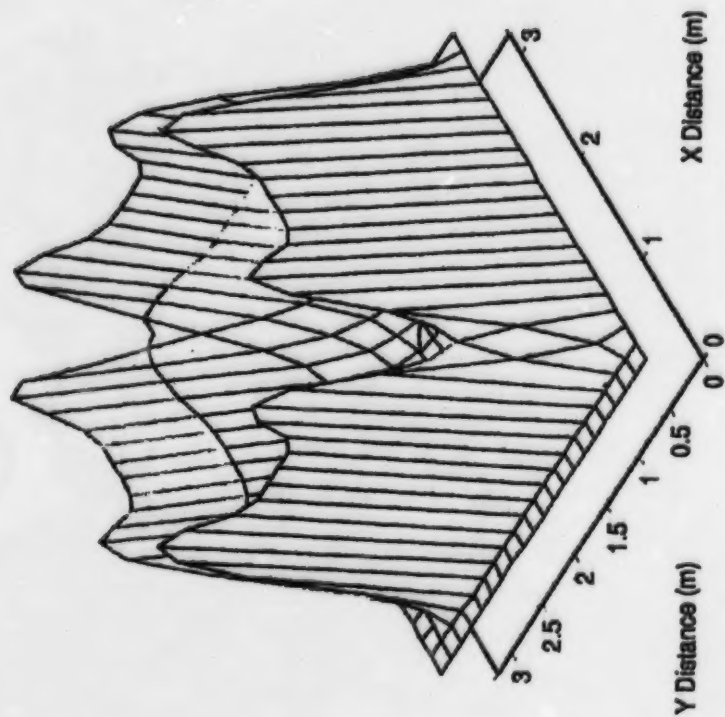


Figure 1

Is-dyna

box2 TNT Time 0.300 (ms) (X-Y) Z = 1.520 (m)



Static Pressure (atm)

7.286812e-01

2.575075e+00

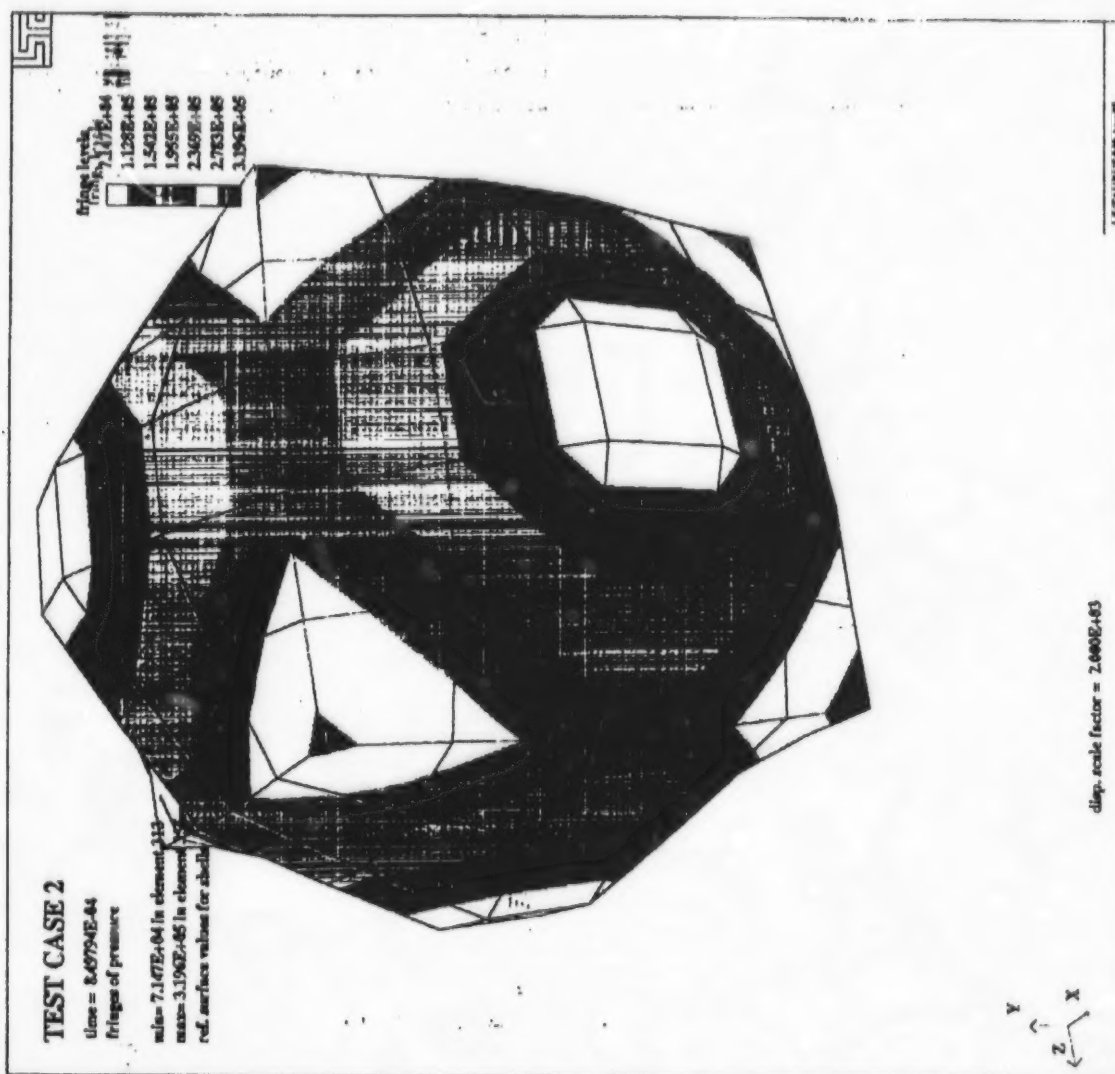
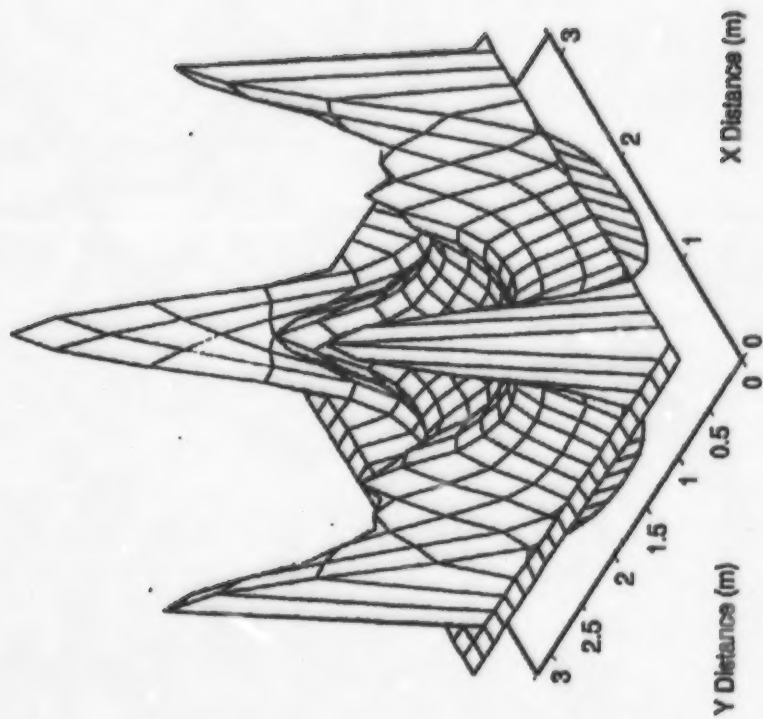


Figure 3

Is-dyna

box2 TNT Time 0.850 (ms) (X-Y) Z = 1.520 (m)



Static Pressure (atm)

7.053155e-01

1.0

2.511681e+00